Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Rok Knez

Oblačna aplikacija za deljenje večopravilnostnih seznamov z ločenim zalednim delom

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Marko Bajec

Ljubljana, 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:
Oblačna aplikacija za deljenje večopravilnostnih seznamov z ločenim zalednim delom

Tematika naloge:

V okviru diplomske naloge razvijte aplikacijo za deljenje večopravilnostnih seznamov s poljubnimi uporabniki. Za razvoj aplikacije uporabite tehnologije, ki omogočajo učinkovit razvoj sodobnih spletnih aplikacij. Razvoj naj zajema dosledno izvedbo analize in načrtovanja pred samim razvojem.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Rok Knez sem avtor diplomskega dela z naslovom:

Oblačna aplikacija za deljenje večopravilnostnih seznamov z ločenim zalednim delom

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marko Bajec,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 2. julija 2015 Podpis avtorja:

Zahvaljujem se vsem svojim bližnjim. Posebej se zahvaljujem svojemu mentorju, prof. dr. Marko Bajecu, za nasvete in pomoč pri izdelavi diplomskega dela.

Vsem, ki so me v življenju usmerjali do trenutne poti. Implicitno ali eksplicitno.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Računalništvo v oblaku **3**

2.1 Modeli storitve 3

2.1.1 Programska oprema kot storitev 3

2.1.2 Platforma kot storitev 4

2.1.3 Infrastruktura kot storitev 4

2.2 Modeli ponudbe 4

2.2.1 Zasebni oblak 4

2.2.2 Družbeni oblak 4

2.2.3 Javni oblaki 5

3 Tehnologije in orodja **7**

3.1 Razvoj uporabniškega vmesnika 7

3.1.1 HTML in HTML5 7

3.1.2 CSS 8

3.1.3 SASS 8

3.1.4 Javascript 8

3.1.5 Bootstrap 9

3.1.6 AngularJS 9

3.1.7 RestAngular . . . 9
3.1.8 ngDialog . . . 10
3.1.9 Yeoman . . . 10
3.1.10 Grunt . . . 10
3.1.11 Bower . . . 10
3.1.12 Sklad tehnologij uporabniškega vmesnika . . . 11
3.2 Zaledni sistem . . . 12
3.2.1 Python . . . 12
3.2.2 Django . . . 12
3.2.3 REST . . . 12
3.2.4 Django REST Framework . . . 13
3.2.5 Sklad tehnologij zalednega sistema . . . 13
3.3 nginx . . . 14
3.4 Virtualenv . . . 14
3.5 GIT . . . 14
3.6 Fabric . . . 15
3.7 PyCharm . . . 15

4 Načrtovanje aplikacije 17
4.1 Ideja . . . 17
4.2 Zahteve in željene funkcionalnosti aplikacije . . . 17
4.3 Specifikacije zahtev aplikacije . . . 18
4.4 Diagram poteka za primer uporabe aplikacije . . . 19
4.4.1 Scenarij primera uporabe: Ustvari seznam . . . 20
4.4.1.1 Pozitiven scenarij . . . 20
4.4.1.2 Alternative scenarija . . . 20
4.4.2 Scenarij primera uporabe: Posodobi seznam . . . 20
4.4.2.1 Pozitiven scenarij . . . 20
4.4.2.2 Alternative scenarija . . . 21
4.4.3 Scenarij primera uporabe: Izbriši seznam . . . 21
4.4.3.1 Pozitiven scenarij . . . 21
4.4.3.2 Alternative scenarija . . . 21

4.4.4 Scenarij primera uporabe: Všečkaj seznam 21
4.4.4.1 Pozitiven scenarij 21
4.4.4.2 Alternative scenarija 22
4.4.5 Scenarij primera uporabe: Kopiraj seznam 22
4.4.5.1 Pozitiven scenarij 22
4.4.5.2 Alternative scenarija 23

5 Razvoj aplikacije **25**
5.1 Zaledni sistem 25
5.1.1 Načrtovanje podatkovne baze 25
5.1.1.1 Entitetno relacijski diagram 26
5.1.1.2 Podatkovni modeli 27
5.1.1.3 Post-procesiranje fotografij seznama 30
5.1.1.4 Kreiranje QR_HASHA 31
5.1.1.5 API 33
5.1.2 Posebni klici 35
5.1.3 Dodatne informacije 36
5.1.4 Iskanje seznamov po značkah 37
5.1.5 Nastavitve strežnika zalednega sistema 38
5.2 Uporabniški vmesnik 40
5.2.1 Datoteke SASS 40
5.2.2 Pogledi 45
5.2.2.1 Priklapljanje nadzornika 45
5.2.2.2 Delo z obrazci 45
5.2.2.3 Pogled za avtomatsko predlaganje seznamov glede na vnos iskalnih parametrov 48
5.2.3 Storitve 49
5.2.3.1 Storitev REST[18] 49
5.2.3.2 Storitev za delo s pojavnimi okni 51
5.2.4 Nadzorniki 52
5.2.4.1 Dodajanje seznama 52

5.2.4.2 Dodajanje novega predmeta posameznega seznama 53
5.2.5 Nadzornik za avtomatsko predlaganje seznamov glede na vnos iskalnih parametrov 56
5.2.6 Nastavitve strežnika uporabniškega vmesnika 58

6 Sklepne ugotovitve **61**

Literatura **63**

A Izgled uporabniškega vmesnika **67**

Seznam uporabljenih kratic

kratica	angleško	slovensko
HTML	Hyper Text Markup Language	označevalni jezik
HTML5	Hyper Text Markup Language v5	nadgrajen HTML
CSS	Cascading Style Sheets	kaskadna slogovna predloga
DOM	Document Object Model	objektni model dokumenta
SASS	Syntactically Awesome Style Sheets	sintaktično super slogovne predloge
REST	Representational state transfer	zastopani prenos stanja
URL	Uniform Resource Locator	enolični krajevnik vira
HTTP	HyperText Transfer Protocol	protokol za prenos hiperpovezav
API	Application Programming Interface	vmesnik za programiranje aplikacij
PIL	Python Imaging Library	Python knjižnica za delo s fotografijami

Povzetek

Vsem se je že zgodilo, da smo v določenem dnevu imeli ogromno opravkov. Preobremenjenost zaradi slabe organiziranosti nas je večkrat pripeljala do neproduktivnih dni, oziroma slabo izpeljanih postopkov. Tako se vsi poslužujemo pisanja večopravilnostnih seznamov, s pomočjo katerih si razdelamo naše naloge in na podlagi teh ustvarimo strukturirano reševanje problemov.

Namen diplomskega dela je bilo razviti aplikacijo za deljenje večopravilnostnih seznamov. Aplikacija omogoča pisanje večopravilnostnih seznamov, iskanje seznamov po značkah, všečkanje seznamov, kopiranje seznamov, komentiranje posameznih seznamov in pregled obstoječih seznamov na strani uporabniškega profila.

Ključne besede: večopravilnostni seznam, spletna aplikacija, AngularJS, Javascript, REST, Django, Python.

Abstract

We have all had days, where we wanted to finish multiple errands in a single day. Feeling overwhelmed from lack of organization, we have all experienced a lack of productivity and poorly finished errands. To help us better achieve and manage daily tasks, we have all, at least once, created a multitasking list to help us master our tasks.

The aim of this thesis was to develop an application to share multitasking lists. The application enables us to create, share, like, copy, comment and view existing lists.

Keywords: multitasking list, web application, AngularJS, Javascript, REST, Django, Python.

Poglavje 1

Uvod

Vsem se je že zgodilo, da smo imeli v določenem dnevu ogromno opravkov. Preobremenjenost zaradi neorganizacije, ali slabe organizacije nas je večkrat pripeljala do neproduktivnih dni, oziroma slabo izpeljanih postopkov. Tako se vsi poslužujemo pisanja večopravilnostnih seznamov, s pomočjo katerih si razdelamo naše naloge in na podlagi teh ustvarimo strukturirano reševanje problemov.

Pri izdelavi naše aplikacije smo si postavili naslednje vprašanje:

Ali se določene naloge ponavljajo do takšne mere, da si lahko uporabniki večopravilnostne sezname delijo?

V sklopu diplomske naloge je bila razvita oblačna aplikacija z ločenim zalednim sistemom, kateri omogoča naknaden priklop mobilnih aplikacij na že obstoječo podatkovno bazo. V aplikaciji uporabnikom ponujamo funkcionalnosti, kot so pisanje večopravilnostnih seznamov, iskanje seznamov po značkah, všečkanje seznamov, kopiranje seznamov, komentiranje posameznih seznamov in pregled obstoječih seznamov na strani uporabniškega profila. Uporabnik lahko posamezen večopravilnostni seznam všečka, ob všečkanju pa se mu seznam prikaže na njegovem profilu. Ob akciji všečkanja se upo-

rabniku na dotičnem seznamu prikažejo potrditvena polja, s pomočjo katerih lahko nato zabeleži, ali je posamezen predmet seznama končal.

Poglavje 2

Računalništvo v oblaku

Računalništvo v oblaku je model za omogočanje vseprisotnega, priročnega dostopa do omrežja na zahtevo, ki pridobiva svoje zmožnosti iz bazena nastavljivih virov za računanje (npr. strežnikov, naprav za shranjevanje, aplikacij, storitev, ipd.).[15]

Razvijalci z inovativnimi idejami za nove internetne storitve tako ne potrebujejo več velike količine kapitala za nakup strojne opreme, ki bi poganjala njihove storitve. Podjetja z veliko serijo računskih nalog lahko s pomočjo računalništva v oblaku pridobijo rezultate tako hitro, kot lahko se lahko njihovi programi širijo. Za primer, uporaba 1000 strežnikov za eno uro stane približno isto, kot 1 strežnik 1000 ur. [16]

2.1 Modeli storitve

2.1.1 Programska oprema kot storitev

Programska oprema kot storitev, ali angleško *Software as a Service (SaaS)* omogoča potrošniku uporabo ponudnikovih aplikacij, katere tečejo na oblačni infrastrukturi. Te aplikacije so dostopne iz različnih klientov, kot je naprimer spletni brskalnik, različne mobilne naprave, ali programski klienti.

2.1.2 Platforma kot storitev

Platforma kot storitev, ali angleško *Platform as a Service (PaaS)* omogoča na strežniško infrastrukturo potrošniku nalaganje in poganjanje lastno ustvarjenih ali pridobljenih aplikacij, kot so naprimer različni skupki programske kode, knjižnic, storitev in orodij, katere podpira ponudnik.

2.1.3 Infrastruktura kot storitev

Infrastruktura kot storitev, ali angleško *Infrastructure as a Service (IaaS)* omogoča potrošniku nadzor, procesiranje, shranjevanje in ostale temeljne računalniške storitve. Potrošnik lahko s pomočjo Infrastrukture kot storitev nalaga in poganja bistveno programsko opremo, kot je naprimer operacijski sistem.

2.2 Modeli ponudbe

2.2.1 Zasebni oblak

Dostop do oblačne infrastrukture je omogočen samo eni pravni ali fizični osebi. Seveda lahko pravno osebo razdelimo na več porabnikov, ki si delijo dostop do oblačne infrastrukture. Urejanje in lastništvo so v celoti prenešene na lastnika, ki lahko upravlja osebno, torej upravljanje prevzame oseba znotraj podjetja, ali pa najame skrbnika za upravljanje z infrastrukturo.

2.2.2 Družbeni oblak

Dostop do oblačne infrastrukture je omogočen posebej določeni skupini potrošnikov iz različnih organizacij, ki imajo skupne cilje (npr. specifične zahteve varnosti). Lastništvo in urejanje oblačne infrastrukture je lahko deljeno na različne člane iz različnih organizacij.

2.2.3 Javni oblaki

Dostop do oblačne infrastrukture je namenjen širši javnosti. Tako lahko z oblačno infrastrukturo upravljajo podjetja, akademske ustanove, javne ustanove, ali kombinacija večih.

Poglavje 3

Tehnologije in orodja

3.1 Razvoj uporabniškega vmesnika

Ker želimo dobro orisati ločen razvoj uporabniškega vmesnika in razvoj zalednega sistema, bomo posebej opisali tehnologije uporabljene za razvoj uporabniškega vmesnika in tehnologije uporabljene za razvoj zalednega sistema.

3.1.1 HTML in HTML5

HTML[1] je označevalni jezik, katerega zgodovina sega v konec leta 1990, ko je Tim Berners-Lee predlagal poenoten sistem za izmenjavo sporočil preko spleta. Uporablja se za strukturiranje in predstavitev vsebine na svetovnem spletu. S pomočjo HTML značk postavimo strukturo spletne strani, katero brskalnik interpretira in prikaže na zaslon. V večini primerov se HTML značke uporabljajo v parih, ki prikažejo začetek in konec vsebine med določeno značko. Primer je značka za poglavje, katere začetek označimo z `<h1>`, konec pa z `</h1>`.

HTML5[2] je nadgradnja tehnologije HTML, poudarek pa temelji predvsem na semantičnih značkah. Tako lahko zamenjamo HTML značke, kot so naprimer `<div> </div>` s semantično vrednejšimi značkami, kot so naprimer `<header> </header>`, `<nav> </nav>`, `<section> </section>`, `<article> </article>`, `<footer> </footer`. Te značke pripomorejo k boljšemu inde-

ksiranju naše spletne strani, saj lahko zaradi povečane semantične vrednosti brskalniki lažje ugotovijo vsebino naše spletne strani. HTML5 nadgradnja vsebuje tudi elemente za predvajanje multimedije `<audio>`, `<video>`, različne grafične elemente, kot sta naprimer `<svg>` in `<canvas>`, katera nam omogočata izris vektorskih grafik.

3.1.2 CSS

Iz potrebe po boljši strukturiranosti, lažjem urejanju in možnosti reciklaže uporabljenih stilov za oblikovanje HTML spletnih strani se je razvil CSS[3]. S pomočjo CSSja lahko uredimo izgled HTML kode z določanjem atributov `class` ali `id` HTML značkam. Te atribute imenujemo tudi označevalniki stilov. Z ločevanjem vsebine in oblikovanja strani je možna ponovna uporaba CSS kode na različnih HTML straneh.

3.1.3 SASS

SASS[4] je skriptni jezik, ki se je razvil zaradi boljše organizacije jezika CSS[3]. Ker je pri kompleksnejših spletnih straneh, kot so naprimer spletne oblačne aplikacije kar nekaj kaskadnih slogov, nam pomaga pri organiziranosti CSS kode SASS. SASS je predprocesor CSSja, ki nam omogoča uporabo spremenljivk, gnezdenih stilov, funkcij in še mnogo drugih dodatkov, ki nam opazno pohitrijo razvoj. Deluje tako, da nam SASS prevajalnik prevede kodo v navaden CSS, katerega lahko potem brskalniki interpretirajo.

3.1.4 Javascript

Javascript[5] je skriptni programski jezik s podporo koncepta objektnega programiranja. Največkrat se uporablja kot podpora HTML[1] kodi za manipulacijo z DOM drevesom. Javascript se izvaja lokalno na klientovem spletnem brskalniku. Struktura Javascripta nam omogoča asinhrono izvajanje funkcij na strani klienta v povezavi s strežnikom. Javascript v projekt vključimo kot zunanjo datoteko, ali pa ga vključimo kar med HTML[1] kodo.

3.1.5 Bootstrap

Izdelava vizualno privlačne spletne strani je lahko zamuden proces. Za lažji in hitrejši razvoj uporabniškega vmesnika se je pojavilo kar nekaj ogrodij. Eno izmed teh ogrodij je ogrodje Bootstrap[6], katerega je razvil Twitter. Ogrodje Bootstrap je sestavljeno iz treh tehnologij, HTML[1], CSS[3] in Javascript[5]. S pomočjo vseh treh tehnologij lahko dostopamo do skupka že zgrajenih gradnikov, ki nam prihranijo kar nekaj časa. Tako lahko naprimer uporabimo že vnaprej slogovno urejena vnosna polja, gumbe, tabele, navigacijo in še mnogo drugih osnovnih gradnikov. Ena izmed funkcionalnosti, s katero se Bootstrap ponaša, je uporaba t.i. odzivnih poizvedb, ki nam omogočajo, da s preprosto sintakso določimo obnašanje spletnih strani za različne širine zaslonov. S tem lahko našo spletno stran prilagodimo tako za računalniške zaslone, kot tudi za mobilne zaslone in tablice.

3.1.6 AngularJS

AngularJS[7] je strukturno ogrodje za razvoj asinhronih dinamičnih spletnih aplikacij. Omogoča razširitev sintakse HTML[1] jezika s svojimi gradniki. Ti gradniki omogočajo dvosmerno vezavo spremenljivk, s katero lahko preprosto in učinkovito sledimo uporabnikovim dogodkom znotraj spletne aplikacije. Zgrajen je na osnovi Javascripta[5], kar pomeni, da se AngularJS aplikacije prevajajo v celoti na strani klienta in s tem manj obremenjujejo strežnik. S pomočjo funkcij manipulira z DOM elementi. Izkorišča asinhrone poizvedbe s t.i. obljubami (promises), katere se izpolnejo takrat, ko dobimo od strežnika odgovor na našo poizvedbo.

3.1.7 RestAngular

Restangular[8] je zunanja knjižnica za AngularJS[7], ki nam olajša delo z zahtevami, kot so `GET, POST, PUT, DELETE, ipd.`. Knjižnica je v AngularJS implementirana v obliki storitve (Service), ki nam omogoča delo z minimalno količino dodatne kode.

3.1.8 ngDialog

Pri izdelavi dobre aplikacije je potrebno razmišljati tudi na uporabniško izkušnjo. Za dobro uporabniško izkušnjo lahko uporabimo pojavna okna. Za delo s pojavnimi okni se v AngularJS[7] uporablja knjižnica ngDialog[9], ki nam olajša in poenoti delo z odpiranjem in zapiranjem posameznih pojavnih oken.

3.1.9 Yeoman

Vzpostavitev novega projekta je lahko časovno potratno opravilo. Zato se je za potrebe vzpostavitve osnovnih praznih projektov razvilo ogrodje Yeoman[10]. Ogrodje je na voljo za veliko različnih orodij in nam vzpostavi osnovno strukturo projekta za željeno ogrodje. Tako lahko s preprostimi ukazi vzpostavljamo tako nov projekt, kot tudi nove gradnike znotraj našega projekta. Nov AngularJS[7] projekt lahko tako ustvarimo z ukazom `yo angular IME_PROJEKTA`, znotraj novega projekta pa lahko ustvarjamo gradnike z ukazi, kot je naprimer `yo angular:view IME_POGLEDA`, ki nam znotraj našega projekta ustvari nov pogled.

3.1.10 Grunt

Pri izgradnji večjih Javascript[5] aplikacij se nam večkrat zgodi, da ponavljamo določene akcije, kot so naprimer prevajanje različnih skupkov kode, pomanjševanje Javascript kode in konkatenacija datotek. Grunt[11] nam te akcije avtomatizira s pomočjo napisane Gruntfile datoteke.

3.1.11 Bower

Bower[12] je orodje za manipulacijo z ogrodji, knjižnicami, sredstvi in pripomočki, katere vključimo v našo spletno aplikacjo. Z njegovo pomočjo lahko v našo spletno aplikacijo vključujemo poljubne pakete, Bower pa nam bo poskrbel za pravilno namestitev in vključitev kode v našo spletno aplikacijo. S

pomočjo datoteke `bower.json` povemo Bowerju katere pakete želimo imeti nameščene, katere nato namestimo z ukazom `bower install`

3.1.12 Sklad tehnologij uporabniškega vmesnika

Za lažjo predstavo povezovanja posameznih tehnologij prilagamo še sklad tehnologij, uporabljenih pri razvoju uporabniškega vmesnika.

Slika 3.1: Sklad tehnologij uporabniškega vmesnika

3.2 Zaledni sistem

3.2.1 Python

Python[13] je objektno usmerjeni, visoko nivojski programski jezik z dinamično semantiko. Jasna in pregledna sintaksa nam omogoča hiter razvoj skalabilnih aplikacij. Glavni poudarek Python programskega jezika je berljivost kode. S pomočjo Pythona lahko programiramo tako objektno usmerjeno, funkcijsko, kot tudi proceduralno. Python se drži par preprostih filozofij za ohranjanje svoje sintakse:

- Lepo je bolje kot grdo,
- Eksplicitno je bolje kot implicitno,
- Preprosto je bolje kot kompleksno,
- Berljivost šteje.

3.2.2 Django

Django[17] je visokonivojsko ogrodje za razvoj spletnih aplikacij, ki temelji na programskem jeziku Python[13]. Razvit je bil nekje jeseni leta 2003, ko so spletni programerji Lawrence Journal-World časopisnega uredništva pričeli z uporabo pythona za razvoj aplikacij. Glavni namen ogrodja Django je pospeševanje razvoja tako preprostih, kot tudi kompleksnejših spletnih aplikacij. Drži se strukutre model-pogled-nadzornik in nam s tem omogoča strukturiran in jasen razvoj.

3.2.3 REST

REST[18] je predlagana arhitektura, kjer je vsak posamezen entitetni vir baze predstavljen z enovitim URL naslovom. Opredeljuje, kako uporabljati že obstoječe standarde znotraj HTTP protokola s pomočjo standardnih HTTP metod kot so `GET, POST, PUT, DELETE, ipd.`. Vsak vir REST storitve

je samoopisen in brez stanja. Do virov lahko dostopamo preko URL naslovov, npr. http://spletnastran.com/seznami/, ali dostop do posameznega vira http://spletnastran.com/seznami/ID_SEZNAMA/. Deluje na podoben način, kot spletni brskalniki, ki s pomočjo zahtev pridobivajo vire spletnih strani. REST protokol omogoča izdelavo skalabilnih aplikacij, saj zaradi brezstanjskih klicev strežnikov ne prekomerno obremenjujemo.

3.2.4 Django REST Framework

Django REST Framework[19] je ogrodje, ki temelji na Djangu[17] in REST[18] arhitekturi. Je zmogljivo in fleksibilno ogrodje, ki omogoča preprosto izgradnjo spletnih APIjev s pomočjo programskega jezika Python[13]. Močno se osredotoča na razvoj aplikacij s pomočjo koncepta model-pogled-nadzornik in nam zaradi tega omogoča izgradnjo preglednih, učinkovitih in skalabilnih REST storitev.

3.2.5 Sklad tehnologij zalednega sistema

Za lažjo predstavo povezovanja posameznih tehnologij prilagamo še sklad tehnologij, uporabljenih pri razvoju zalednega sistemaa.

Slika 3.2: Sklad tehnologij zalednega sistema

3.3 nginx

Spletni strežniki so posebni računalniki, ki nam dostavljajo spletne strani. Vsak spletni strežnik je predstavljen z določenim IP naslovom in verjetno tudi z domenskim imenom. Za strežniške storitve obstaja že kar nekaj rešitev, za naše potrebe pa je najprimernejši nginx[20] spletni strežnik, saj nam omogoča visoko zmogljivost z nizko porabo spomina. Nginx se, za razliko od največkrat uporabljenega strežniškega sistema Apache[21], uporablja predvsem za aplikacije, ki imajo veliko hkratnih povezav na strežnik[22]. Apache za vsako povezavo ustvarja nove procese in niti. Tako lahko pri Apache strežniku nastavimo maksimalno število dovoljenih procesov. To lahko pripelje do izstradanja računalniških sredstev, zato se pri večjih sistemih Apache obnese slabo. Nginx, za razliko od Apacheja teče na nekaj, prednastavljenih delavskih procesih, ki jih dodeli upravljalec strežnika.

3.4 Virtualenv

Pri razvoju aplikacije velikokrat potrebujemo knjižnice in orodja, ki so specifična za projekt. Zaradi tega si ne želimo smetiti našega globalnega razvojnega okolja, saj bi lahko to prineslo neprijetne posledice, kot je naprimer razvoj enega projekta v Pythonu 3 (in inštalacija pripadajočih knjižnic), drugega pa v Pythonu 2.7. Ta problem rešimo s posebnim orodjem, ki se mu reče Virtualenv[23]. Virtualenv je orodje, ki nam ustvari navidezno razvojno okolje za Python[13] projekte, kamor lahko prosto nalagamo željene knjižnice in ogrodja specifična za projekt.

3.5 GIT

GIT[24] je orodje za porazdeljeno kontroliranje izvorne kode. Omogoča nam sočasen razvoj aplikacij na istih datotekah z več različnimi razvijalci brez prepisovanja in izginjanja delov kode. Razvoj GITa se je pričel Aprila 2005, po tem, ko je več razvijalev Linux jedra opustilo dostop do BitKeeprja, ki je tudi

orodje za kontroliranje izvorne kode. S pomočjo ukazov, kot so `git pull`, `git push`, `git add`, `git commit` in še drugimi nam omogoča kontroliranje izvorne kode in sočasen razvoj projektov.

3.6 Fabric

Fabric[25] je Python[13] knjižnica, s pomočjo katere si pospešimo in olajšamo prenos lokalnih datotek in razvojnih okolij na strežnik. S pomočjo vnaprej napisanih funkcij lahko večkrat ponovljive dogodke, kot so naprimer inštalacija navideznega okolja, konfiguracija nginx[20] strežnika, inštalacija Djanga[17] in prenos celotnega lokalnega projekta napišemo samo enkrat, za ponovitev pa kasneje poganjamo le funkcije.

3.7 PyCharm

PyCharm[26] je razvojno okolje, ki ima podporo za programiranje Python[13], Javascript[5], HTML[1], CSS[3],... PyCharm nam olajša razvoj aplikacij s podporo za navidezna okolja, vgrajeno konzolo, predlagalnikom in zaključevalnikom sintakse, barvanjem sintakse in še čim. Napisan je v programskem jeziku Java in Python.

Poglavje 4

Načrtovanje aplikacije

4.1 Ideja

Ideja o izdelavi aplikacije se je porodila na podlagi pregleda že obstoječih spletnih in mobilnih aplikacij za deljenje večopravilnostnih seznamov. Vse obstoječe rešitve, kot so naprimer Wunderlist[32], Todoist[33] in ANY.DO[34], omogočajo dodajanje, urejanje in pregled lastnih seznamov, nobena od naštetih aplikacij pa ne omogoča željene fukncionalnosti iskanja po javnih seznamih. Tako smo se po krajšem premisleku odločili za razvoj aplikacije, ki bo omogočala deljenje večopravilnostnih seznamov.

4.2 Zahteve in željene funkcionalnosti aplikacije

Glavni cilj je razviti tako ločen zaledni sistem aplikacije, ki bo omogočal priklop poljubnih odjemalcev, kot tudi spletni uporabniški vmesnik aplikacije. Cilj je omogočiti dodajanje, urejanje in brisanje seznamov. Znotraj posameznega seznama želimo omogočiti osnovne naloge, kot so dodajanje, urejanje in brisanje predmetov, kot tudi nastavljanje fotografije posameznega seznama in dodajanje značk seznama, po katerih bo omogočeno tudi iskanje.

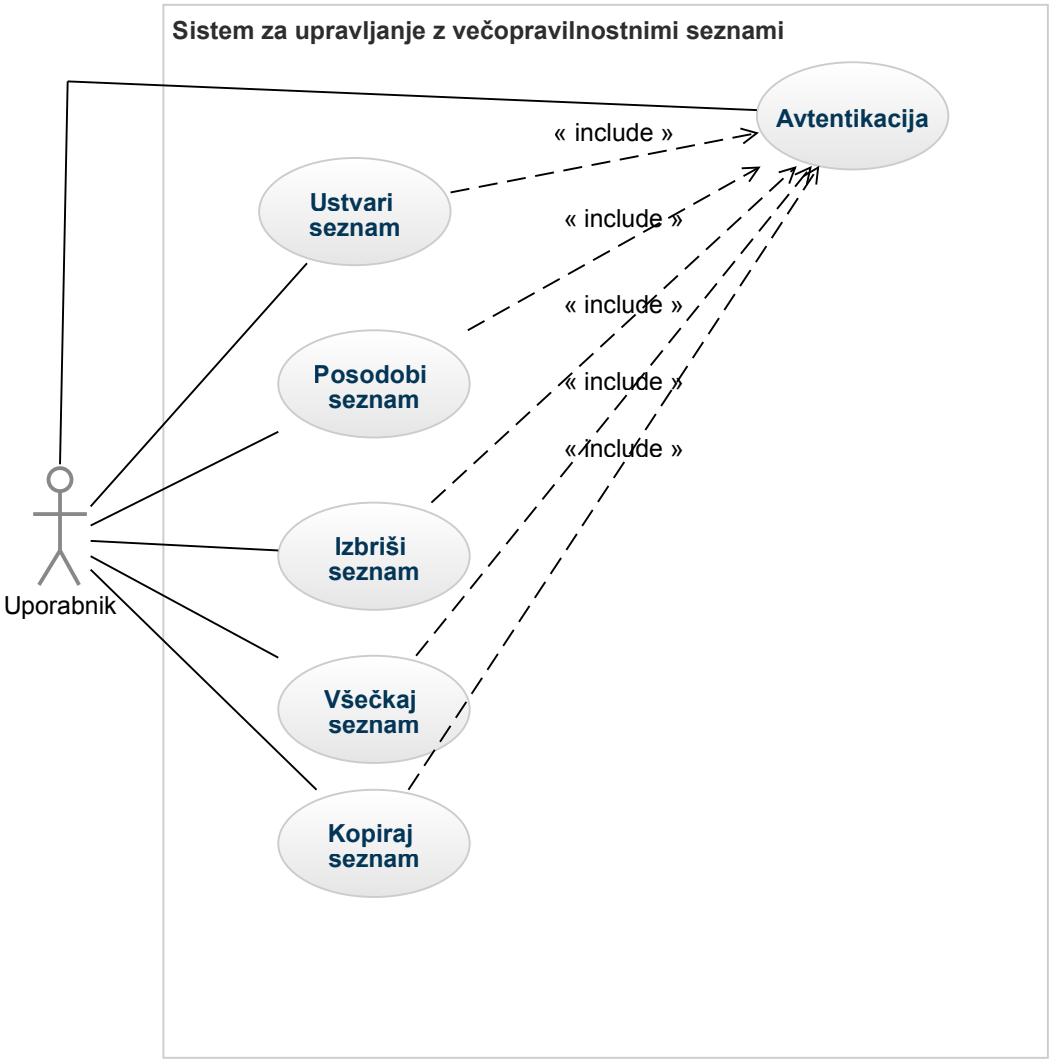
4.3 Specifikacije zahtev aplikacije

Glede na opisane zahteve podajamo specifikacije funkcionalnosti aplikacije:

- aplikacija naj ima ločen zaledni sistem, ki omogoča preprost priklop poljubnih odjemalcev (spletnih, mobilnih, ...),
- za hrambo podatkov naj uporablja PostgreSQL bazo podatkov,
- aplikacija naj uporabniku omogoča brezplačno hrambo seznamov,
- aplikacija naj omogoča osnovne akcije večopravilnostnih seznamov,
- aplikacija naj omogoča deljenje in iskanje seznamov,
- aplikacija naj omogoča iskanje po značkah, pripete posameznemu seznamu,
- aplikacija naj omogoča všečkanje in kopiranje javnih seznamov,
- aplikacija naj omogoča javni prikaz profila registriranega uporabnika in njegovih ustvarjenih seznamov

4.4 Diagram poteka za primer uporabe aplikacije

Za lažjo predstavo osnovnih zahtev prilagamo diagram primerov uporabe.



Slika 4.1: Diagram primerov uporabe

4.4.1 Scenarij primera uporabe: Ustvari seznam

4.4.1.1 Pozitiven scenarij

- Uporabnik stisne gumb "Ustvari seznam"
- Sistem ugotovi, da uporabnik ni prijavljen
- Sistem zahteva vnos uporabniškega imena in gesla (uporabniško ime in geslo sta pravilna)
- Odpre se vnosno okno za dodajanje novega seznama
- Uporabnik izpolni polja in doda nov seznam

4.4.1.2 Alternative scenarija

- Uporabnik se ne uspe avtenticirati
- Odpre se vnosno okno za prijavo
- Uporabnik prekine akcijo s pritiskom na ustrezen gumb

4.4.2 Scenarij primera uporabe: Posodobi seznam

4.4.2.1 Pozitiven scenarij

- Uporabnik poišče seznam, katerega želi posodobiti
- Sistem zahteva vnos uporabniškega imena in gesla (uporabniško ime in geslo sta pravilna)
- Uporabnik izvede akcijo, katerega posledica je prikaz vnosnih polj za posodabljanje seznama
- Uporabnik posodobi seznam in shrani spremembe

4.4.2.2 Alternative scenarija

- Uporabnik se ne uspe avtenticirati
- Odpre se vnosno okno za prijavo
- Uporabnik prekine akcijo s pritiskom na ustrezen gumb ali pa se prijavi in nadaljuje urejanje seznama

4.4.3 Scenarij primera uporabe: Izbriši seznam

4.4.3.1 Pozitiven scenarij

- Uporabnik poišče seznam, katerega želi izbrisat
- Sistem zahteva vnos uporabniškega imena in gesla (uporabniško ime in geslo sta pravilna)
- Uporabnik izvede akcijo brisanja
- Uporabnik izbriše seznam
- Sistem uporabnika preusmeri na spisek obstoječih seznamov

4.4.3.2 Alternative scenarija

- Uporabnik se ne uspe avtenticirati
- Odpre se vnosno okno za prijavo
- Uporabnik prekine akcijo s pritiskom na ustrezen gumb ali pa se prijavi in konča akcijo brisanja

4.4.4 Scenarij primera uporabe: Všečkaj seznam

4.4.4.1 Pozitiven scenarij

- Uporabnik poišče seznam

- Sistem zahteva vnos uporabniškega imena in gesla (uporabniško ime in geslo sta pravilna)
- Uporabnik izvede akcijo všečkanja
- Sistem mu zabeleži akcijo všečkanja in prenese seznam med njegove obstoječe sezname
- Sistem obvesti uporabnika o uspešno izvedeni akciji

4.4.4.2 Alternative scenarija

- Uporabnik se ne uspe avtenticirati
- Odpre se vnosno okno za prijavo
- Uporabnik prekine akcijo s pritiskom na ustrezen gumb ali pa se prijavi in konča akcijo všečkanja

4.4.5 Scenarij primera uporabe: Kopiraj seznam

4.4.5.1 Pozitiven scenarij

- Uporabnik poišče seznam
- Sistem zahteva vnos uporabniškega imena in gesla (uporabniško ime in geslo sta pravilna)
- Uporabnik izvede akcijo kopiranja
- Sistem mu zabeleži akcijo kopiranja in prenese seznam med njegove obstoječe sezname
- Sistem obvesti uporabnika o uspešno izvedeni akciji

4.4.5.2 Alternative scenarija

- Uporabnik se ne uspe avtenticirati
- Odpre se vnosno okno za prijavo
- Uporabnik prekine akcijo s pritiskom na ustrezen gumb ali pa se prijavi in konča akcijo kopiranja

Poglavje 5

Razvoj aplikacije

Razvoj zalednega sistema aplikacije je potekal ločeno od razvoja uporabniškega vmesnika. Za ločitev smo se odločili tako zaradi organiziranosti kode, kot tudi zaradi dejstva, da lahko na podatkovni vir naknadno priključimo še druge odjemalce, kot so naprimer mobilne naprave, brez da vplivamo na uporabniški vmesnik spletne aplikacije.

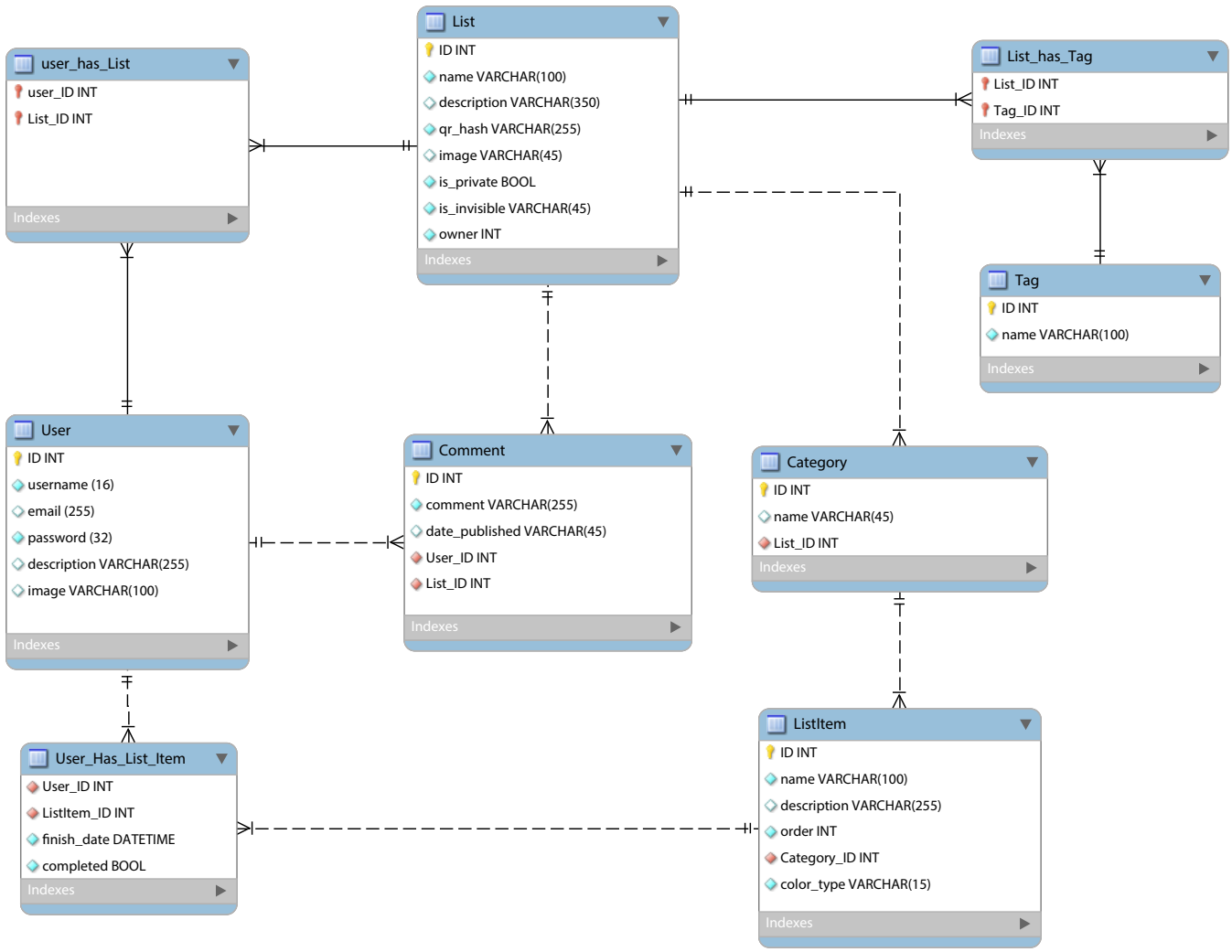
5.1 Zaledni sistem

5.1.1 Načrtovanje podatkovne baze

Struktura baze je bila v obliki entitetno relacijskega diagrama zasnovana pred samim pričetkom razvoja, saj smo s tem lažje planirali tako postavitev samega grafičnega vmesnika, kot tudi same funkcionalnosti aplikacije.

5.1.1.1 Entitetno relacijski diagram

Strukturo baze smo predstavili s pomočjo entitetno relacijskega diagrama. Zasnovana je s pomočjo osnovnih relacij med objekti.



Slika 5.1: Struktura zalednega sistema aplikacije sharely.org

Iz diagrama so razvidne naslednje relacije:

- vsakemu seznamu (List) lahko pripada več uporabnikov (User), vsak uporabnik (User) lahko pripada večim seznamom (List),
- vsakemu seznam (List) pripada več kategorij (Category), ena kategorija (Category) pripada samo enemu seznamu (List),
- vsak predmet (ListItem) lahko pripada samo eni kategoriji (Category), vsaki kategoriji pripada več predmetov (ListItem),
- vsakemu seznamu pripada več značk (Tag),
- vsak uporabnik (User) lahko brez podvojevanja seznamov (List), kategorij (Category) in predmetov (ListItem) označuje predmete (ListItem) končane in nedokončane. To smo dosegli z izdelavo vmesne tabele User_Has_List_Item

5.1.1.2 Podatkovni modeli

Razvoj APIja v Django Rest Frameworku[19] sledi konceptu model-pogled-nadzornik. Zato smo za vsako entiteto morali pripraviti lasten model, ki deduje iz razreda `models.Model`. Spodaj bomo prikazali uporabo na nekaj primerih.

Seznam (List)

Za podatkovni model seznama (List) smo glede na zastavljen entitetno relacijski diagram pripravili naslednjo kodo:

```
1 from django.db import models
2 from django.contrib.auth.models import User
3 from apps.Tag.models import Tag
4
5 class List(models.Model):
6     id = models.AutoField(primary_key=True, blank=False)
```

```
7     name = models.CharField(max_length=100, blank=False,
          default='')
8     qr_hash = models.CharField(max_length=255, blank=False,
           default='', unique=True)
9     description = models.TextField()
10    header_img_url = models.ImageField(blank=True, null=
          True)
11    is_private = models.BooleanField(blank=False, default=
          False)
12    is_invisible = models.BooleanField(blank=False, default
          =False)
13    owner = models.ForeignKey(User, related_name='
          list_owner', default=1)
14    users = models.ManyToManyField(User, related_name='
          list_users', blank=True)
15    tags = models.ManyToManyField(Tag, related_name='
          list_tags', blank=True)
```

Razvidno je, da smo vezali lastnika (`owner`) posameznega seznama (List) s tujim ključem na privzeti Django[17] model uporanika (User). Ker bo funkcionalnost uporabniškega vmesnika omogočala uporabnikom všečkanje posameznih seznamov, smo ustvarili polje `users`, kateremu smo določili obojesmerno razmerje (ManyToMany). To pomeni, da lahko ima en uporabnik (User) več seznamov (List) in en seznam več uporabnikov. Isto smo storili še s poljem `tags`.

Kategorija (Category) Za podatkovni model kategorije (Category) smo glede na zastavljen entitetno relacijski diagram pripravili naslednjo kodo:

```
1 from django.db import models
2 from apps.List.models import List
3
4 class Category(models.Model):
5     id = models.AutoField(primary_key=True, blank=False)
```

```
6     name = models.CharField(max_length=100, blank=False,
          default='')
7     list = models.ForeignKey(List, related_name='
          list_categories', default=0)
```

Predmet (ListItem) Za podatkovni model predmeta (ListItem) smo glede na zastavljen entitetno relacijski diagram pripravili naslednjo kodo:

```
1 from datetime import datetime
2 from django.contrib.auth.models import User
3 from django.db import models
4 from apps.Category.models import Category
5
6 class ListItem(models.Model):
7     id = models.AutoField(primary_key=True, blank=False)
8     name = models.CharField(max_length=100, blank=False,
          default='')
9     description = models.TextField(max_length=300, blank=
          True)
10    order = models.IntegerField(blank=False, default=0)
11    color_type = models.CharField(choices=COLOR_TYPES,
          default='color-one', max_length=11)
12    category = models.ForeignKey(Category, related_name='
          list_items', default=0)
13    users = models.ManyToManyField(User, related_name='
          list_item_users', default=1, through='
          UserHasListItem')
```

Pri polju `users` smo posebej določili značko `through`, ki nam pove skozi katero dodatno tabelo spremljamo relacijo uporabnik-predmet. Ker smo želeli beležiti dodatna polja, torej datum predmeta in stanje (končano/nedokončano), smo pripravili še dodatno vmesno tabelo.

Za pripravo vmesne tabele **UserHasListItem** smo uporabili sledečo kodo:

```
1 from django.db import models
2 from apps.List.models import List
3
4 class UserHasListItem(models.Model):
5     user = models.ForeignKey(User)
6     list_item = models.ForeignKey('ListItem')
7     finish_date = models.DateTimeField('finish date', blank
          =False, default=datetime.now)
8     completed = models.BooleanField(blank=False)
```

5.1.1.3 Post-procesiranje fotografij seznama

Ker preko polja `header_img_url` omogočamo nalaganje fotografij, je bilo potrebno zagotoviti učinkovito hranjenje fotografij. Za to smo uporabili Python[13] knjižnico PIL[14], s pomočjo katere bomo naloženim fotografijam spremenili velikost in kvaliteto.

Django rest framework zaradi koncepta dedovanja omogoča prepisovanje privzetih metod. Če želimo ujeti klic metode, kjer shranimo model, to storimo s sledečo kodo:

```
1 from PIL import Image
2
3 class List(models.Model):
4
5     ...
6
7     def save(self, size=(1280, 1280), **kwargs):
8         super(List, self).save()
9
10        if not self.header_img_url:
11            return
12
13        filename = self.header_img_url.path
14        image = Image.open(filename)
```

```
15
16          image.save(filename, format='JPEG', quality=80)
```

5.1.1.4 Kreiranje QR_HASHA

Vsak seznam naslavljamo preko dodeljenega QR_HASHA. Ker je bilo potrebno ob dodajanju novega seznama ustvariti naključen QR_HASH, smo za to napisali posebno metodo.

Metoda za svoje delovanje potrebuje knjižnico `get_random_string` iz zbirke `django.utils.crypto`.

Koda nam prikaže prepis dedovane metode `create()`, kjer kličemo najprej starševsko metodo `create()`, s pomočjo katere ustvarimo objekt, nato pa novemu objektu dodelimo naključen QR_HASH:

```
1  from django.utils.crypto import get_random_string
2
3  class ListSerializer(serializers.ModelSerializer):
4      ...
5
6      def create(self, validated_data):
7
8              ...
9
10             list = super(ListSerializer, self).create(
                   validated_data)
11             list.qr_hash = self.random_hash()
12
13             ...
14
15             list.save()
16
17             return list
18
19     @staticmethod
20     def random_hash():
```

```
21
22          random_string = get_random_string(length=5)
23
24          # Check if hash exists
25          num_results = List.objects.filter(qr_hash=
                random_string).count()
26
27          if num_results > 0:
28              return ListSerializer.random_hash()
29
30          return random_string
```

Iz kode je razvidno, da ustvarjamo naključen niz dolžine 5 znakov. V primeru, da QR_HASH že obstaja, s pomočjo rekurzije niz ponovno ustvarimo.

Knjižnica `crypto` vzame za osnovo angleško abecedo. Če želimo narediti izračun števila možnih kombinacij, moramo upoštevati naslednje parametre:

- 26 velikih znakov (A-Z)
- 26 majhnih znakov (a-z)
- 10 števil (0-9)

Skupaj torej 56 možnih znakov. Za izračun števila možnih kombinacij vzamemo formulo za variacije s ponavljanjem[27]:

Izrek 5.1 *Število variacij n elementov na r prostih mest je enako:*

$$V = n^r. \tag{5.1}$$

V našo enačbo vstavimo 56 možnih znakov, kateri se lahko poljubno ponavljajo na 5. prostih mestih:

$$V = 56^5 = 550.731.776 \tag{5.2}$$

Z dolžino 5 naključnih znakov lahko torej ustvarimo skoraj 551 milijonov kombinacij in posledično skoraj 551 milijonov unikatnih naslovov za posamezne sezname.

5.1.1.5 API

Uporabniški vmesnik spletne aplikacije dostopa do storitev zalednega sistema. Če želimo dostopati do posameznih entitet nam Django Rest Framework[19] to omogoča z interno knjižnico `routers`.

Uvozimo jo lahko s pomočjo klica:

```
1 from rest_framework import routers
```

Za pripravo posameznih vstopnih točk za dostop do entitet smo pripravili sledečo kodo:

```
1  router = routers.DefaultRouter()
2
3  router.register(r'tags', ListTagViewSet, base_name='tags')
4
5  router.register(r'lists', ListViewSet, base_name='lists')
6  router.register(r'p', PrivateListViewSet)
7  router.register(r'i', InvisibleListViewSet)
8
9  router.register(r'mylists', MyListsViewSet)
10 router.register(r'myprivatelists', MyPrivateListsViewSet)
11 router.register(r'myinvisiblelists',
       MyInvisibleListsViewSet)
12 router.register(r'mylovedlists', MyLovedListsViewSet)
13
14 router.register(r'u', ProfilePublicView)
15
16 router.register(r'toplists', TopListsViewSet)
17 router.register(r'lists/(?P<qr_hash>[^/.]+)/tags',
       TagViewSet, base_name='tags')
18 router.register(r'lists/(?P<qr_hash>[^/.]+)/comments',
       CommentViewSet, base_name='comments')
19 router.register(r'lists/(?P<qr_hash>[^/.]+)/addcomment',
       CreateCommentViewSet, base_name='comments_add')
20 router.register(r'lists/(?P<qr_hash>[^/.]+)/categories',
       CategoryViewSet, base_name='categories')
```

```
21 router.register(r'lists/(?P<qr_hash>[^/.]+)/categories/(?P<
       category_pk>[0-9]+)/listitems', ListItemViewSet,
       base_name='listitems')
```

Kot je razvidno, lahko do posameznih entitet dostopamo s pomočjo naslovov. Primer dostopa bi bil http://api.XYZ.org/api/v1/lists/, kjer dostopamo do vseh entitet objekta seznam (List).

Primer klica posamezne entitete bi bil http://api.XYZ.org/api/v1/lists/QR_HASH in nam vrne sledeči odgovor:

```
1  {
2      "id": 2,
3      "name": "The bucket list",
4      "qr_hash": "iZT61",
5      "request_user_loved_list": false,
6      "owner": {
7          "username": "Klemen",
8          "id": 4
9      },
10     "is_owner": false,
11     "is_private": false,
12     "is_invisible": false,
13     "is_public": true,
14     "description": "This are the things I want to do
           before I die",
15     "list_item_count": 1,
16     "users_loved_count": 0,
17     "list_categories": [
18         {
19             "id": 1,
20             "name": "to do",
21             "list": 2,
22             "list_items": [
23                 {
24                     "id": 1,
25                     "name": "Jump with the parachute",
26                     "order": 0,
```

```
27                      "color_type": "color-two",
28                      "user_has_item_complete": false,
29                      "description": "",
30                      "category": 1
31                  }
32              ]
33          }
34      ],
35      "header_img_url": "http://api.XYZ.org/api/v1/lists/
           iZT61/media/bucketList.jpg",
36      "tags": [
37          {
38              "id": 1,
39              "text": "bucket"
40          },
41          {
42              "id": 2,
43              "text": "list"
44          }
45      ]
46  }
```

5.1.2 Posebni klici

Ker potrebujemo zaradi specifike spletne strani kar nekaj klicov, ki so vezani na zahtevke uporabnika, smo jih posebej pripravili znotraj posameznih pogledov `views.py`.

V Django Rest Frameworku[19] lahko to rešimo s pomočjo interne knjižnice `detail_route`.

Uvozimo jo lahko s pomočjo klica:

```
1 from rest_framework.decorators import detail_route
```

Primer posebnega klica je naprimer všečkanje in odvšečkanje posameznega seznama.

Ti poizvedbi določimo znotraj datoteke `views.py` in sicer s sledečo kodo:

```
1 @detail_route(methods=['POST'])
2 def love(self, request, qr_hash=None):
3     try:
4         query_list = List.objects.get(qr_hash=qr_hash)
5         query_list.users.add(self.request.user)
6 
7         return Response(status=status.HTTP_200_OK)
8     except List.DoesNotExist:
9         return Response(status=status.HTTP_404_NOT_FOUND)
10 
11 @detail_route(methods=['POST'])
12 def unlove(self, request, qr_hash=None):
13     try:
14         query_list = List.objects.get(qr_hash=qr_hash)
15         query_list.users.remove(self.request.user)
16 
17         return Response(status=status.HTTP_200_OK)
18      except List.DoesNotExist:
19         return Response(status=status.HTTP_404_NOT_FOUND)
```

5.1.3 Dodatne informacije

Kar nekaj serializiranih polj se ustvari tekom same poizvedbe in v bazi, oziroma modelih niso shranjena.

Primer takšnih polj so naprimer podatki o tem, ali je uporabnik seznam (List) že všečkal, ali še ne. Za to v Django Rest Frameworku[19] uporabimo poseben tip polja, ki se mu reče `SerializerMethodField()`.

Primer metode, kjer ugotovimo, če je uporabnik seznam (List) že všečkal:

```
def get_request_user_loved_list(self, obj):
    request_user = self.context.get('request').user

    if request_user.is_authenticated():
        user_profile = UserProfile.objects.get(user=
            request_user)

        return List.objects.get(qr_hash=obj.qr_hash).users.
            filter(userprofile=user_profile).exists()

    return False
```

5.1.4 Iskanje seznamov po značkah

Aplikacija nam omogoča iskanje po značkah (Tag), katere so dodane posameznemu seznamu (List). To lahko v Django Rest Frameworku [19] omogočimo s sledečo kodo:

```
def get_queryset(self):

    search_param = self.request.QUERY_PARAMS.getlist('
        search', None)

    if len(search_param) > 0:
        """
        Iterate over params and get queryset
        """
        queryset = []
        tag_id_set = []

        #get all tags by id
        for param in search_param:
            param = param.lower()
            tag, created = Tag.objects.get_or_create(text=
                param)
            tag_id_set.append(tag.id)
```

```
17
18          queryset.extend(List.objects.filter(Q(tags__id__in=
                tag_id_set, is_private=False, is_invisible=False
                ))
19                          .distinct())
20
21          # Check by name also
22          for param in search_param:
23              queryset.extend(
24                  List.objects.filter(Q(name__contains=param,
                        is_private=False, is_invisible=False))
25              )
26
27          return list(set(queryset)) #Remove duplicates -
                turn to set and to array again
28
29      return List.objects.filter(is_private=False,
            is_invisible=False)
```

5.1.5 Nastavitve strežnika zalednega sistema

Ker za serviranje datotek tako zalednega sistema, kot tudi uporabniškega vmesnika uporabljamo nginx[20], je bilo potrebno ustvariti posebno konfiguracijsko datoteko znotraj poti `/etc/nginx/sites-available`. V konfiguracijski datoteki nastavimo pot do zapisnikov, ime strežnika in pot strežnika.

Primer konfiguracijske datoteke zalednega sistema:

```
1 upstream app_server_dev {
2         server 127.0.0.1:8000 fail_timeout=0;
3 }
4
5 log_format formatted_log '$http_x_forwarded_for -
      $remote_user [$time_local] ' '"$request" $status
      $body_bytes_sent "$http_referer" ' '"$http_user_agent"'
      ;
6
```

```
 7 server {
 8         listen 80;
 9         client_max_body_size 4G;
10         server_name ['xx.xxx.xxx.xxx'] api.XYZ.org;
11
12         keepalive_timeout 5;
13
14         access_log  /home/username/zaledniSistem/logs/api.
               access.log formatted_log;
15
16         root /home/username/zaledniSistem;
17
18         location / {
19                 try_files $uri @proxy_to_app;
20
21                 gzip             on;
22                 gzip_min_length  1000;
23                 gzip_proxied     expired no-cache no-store
                       private auth;
24                 gzip_types       text/plain text/xml text/
                       css application/xhtml+xml application/
                       xml application/rss+xml application/
                       javascript application/x-javascript;
25                 gzip_disable     "MSIE [1-6]\.";
26         }
27
28         location @proxy_to_app {
29                 proxy_set_header X-Forwarded-For
                       $proxy_add_x_forwarded_for;
30                 proxy_set_header Host $http_host;
31                 proxy_set_header X-Real-IP $remote_addr;
32                 proxy_set_header REMOTE_HOST $remote_addr;
33                 proxy_set_header X-FORWARDED-PROTOCOL
                       $scheme;
34                 proxy_redirect off;
35
36                 proxy_pass http://app_server_dev;
37         }
```

```
38 }
```

5.2 Uporabniški vmesnik

Za izdelavo uporabniškega vmesnika smo uporabili strukturno ogrodje za razvoj asinhronih dinamičnih spletnih aplikacij AngularJS[7]. AngularJS omogoča razvoj aplikacij s pomočjo koncepta model-pogled-nadzornik, zato je bilo potrebno pripraviti kar nekaj različnih nadzornikov in pogledov. Za pomoč pri večkrat uporabljenih storitvah se pripravijo t.i. storitve (Services), ki nam pomagajo čimbolj optimalno uporabiti in reciklirati večkrat napisano kodo. Pri izgledu posameznih pogledov smo si pomagali s predprocesorjem CSS[3] kode SASS[4].

Za lažjo predstavo izgleda uporabniškega vmesnika smo v dodatek A dodali tudi izgled posameznih pogledov.

5.2.1 Datoteke SASS

Zaradi strukture in organiziranosti datotek s stili v formatu SASS[4], smo se odločili za organizacijo kode, kjer v glavno datoteko `main` uvozimo vse ostale datoteke za podstrani. Na tem mestu je dobro omeniti še, da smo za vse obstoječe datoteke, razen za datoteko `main` pred ime datoteke dodali znak `"_"`. Ta znak v SASSu pomeni, da se datoteke s tem predznakom ne prevajajo, saj želimo, da se prevedejo samo enkrat, ob uvozu v glavno datoteko `main`. Ko ostale datoteke uvozimo v glavno datoteko, lahko ta preznak izpustimo, saj prevajalnik besedam ob uvozu predznak avtomatsko doda.

Primer datoteke s stili `main`:

```
1 //Import font awesome
2 @import url('//maxcdn.bootstrapcdn.com/font-awesome/4.3.0/
      css/font-awesome.min.css');
3
```

```
4  @import "mixins";
5  @import "variables";
6  @import "FiveZeroZero";
7  @import "dialogs";
8  @import "tags";
9  @import "searchbox";
10 @import "header";
11 @import "login";
12 @import "register";
13 @import "profile";
14 @import "common";
15 @import "landing";
16 @import "searchresults";
17 @import "list/list";
18 @import "list/listdetails";
19 @import "list/listqr";
20 @import "list/listconfigure";
21 @import "inputDialog";
22 @import "addlist";
23 @import "tos";
24 @import "privacypolicy";
25 @import "footer";
```

Kot je razvidno iz kode, najprej uvozimo dve posebni datoteki `mixins` in `variables`. Ti dve datoteki vsebujeta naše globalne spremenljivke in funkcije, ki nam olajšajo in skrajšajo delo s stili.

Primer datoteke `variables`:

```
1  @import url('http://fonts.googleapis.com/css?family=Open+
       Sans:400,300,600,700');
2
3  $font-family: 'Open Sans', sans-serif;
4
5  //Border radius
6  $border-radius: 2px;
7
8  //transition time
```

```
9  $transition-time-normal: 0.5s;
10
11 // Colors
12 $background-color: #f0f2f5;
13
14 $brand-primary: #5acbe8;
15 $brand-primary-hover: #3FAEB6;
16
17 $brand-dark: #4B4B4B;
18
19 $brand-success: #D0F487;
20
21 $brand-light-pink: #fb5455;
22 $brand-light-pink-hover: #a43838;
23
24 $brand-gray: #99a3b1;
25 $brand-gray-hover: #707a88;
26
27 $box-shadow-color: #e3e4e8;
28
29 $brand-facebook-color: #5d9cec;
30 $brand-facebook-color-hover: #4b89dc;
31
32 $brand-twitter-color: #4fc0e8;
33 $brand-twitter-color-hover: #3aafda;
34
35 /*
36   List item colors
37 */
38
39 $color-list-yellow: #ffce55;
40 $color-list-yellow-light: #fee39f;
41 $color-list-yellow-text: #a67f1f;
42
43 $color-list-blue: #4fc0e8;
44 $color-list-blue-light: #ceeffb;
45
46 $color-list-green: #a0d468;
```

```
47 $color-list-green-light: #dceacd;
48
49 $color-list-lightpink-light: #fdd4d5
```

Ker se lahko barvna shema tekom razvoja spremeni, je dobro imeti ločene globalno nastavljene spremenljivke, katere lahko spremenimo na enem mestu. To nam olajša delo z barvnimi shemami in drastično zmanjša možnost napak pri vnosu barvnih kod.

Primer datoteke `mixins`:

```
1  /*
2  Sass mixins
3  */
4
5  @mixin border-radius($radius) {
6    -webkit-border-radius: $radius;
7    -moz-border-radius: $radius;
8    -ms-border-radius: $radius;
9    border-radius: $radius;
10 }
11
12 @mixin border-top-left-radius($radius) {
13   -webkit-border-top-left-radius: $radius;
14   -moz-border-top-left-radius: $radius;
15   -ms-border-top-left-radiuss: $radius;
16   border-top-left-radius: $radius;
17 }
18
19 @mixin border-top-right-radius($radius) {
20   -webkit-border-top-right-radius: $radius;
21   -moz-border-top-right-radius: $radius;
22   -ms-border-top-right-radius: $radius;
23   border-top-right-radius: $radius;
24 }
25
26 @mixin box-shadow($params) {
27   -webkit-box-shadow: $params;
```

```
28    -moz-box-shadow: $params;
29    box-shadow: $params;
30  }
31
32  @mixin transform($property) {
33    -webkit-transform: $property;
34    -moz-transform: $property;
35    -ms-transform: $property;
36    -o-transform: $property;
37    transform: $property;
38  }
39
40  @mixin transition($property, $time) {
41    -webkit-transition: $property $time;
42    -moz-transition: $property $time;
43    -ms-transition: $property $time;
44    -o-transition: $property $time;
45    transition: $property $time;
46  }
47
48  @mixin animation($properties) {
49    -moz-animation: $properties;
50    -webkit-animation: $properties;
51    -o-animation: $properties;
52    -ms-animation: $properties;
53    animation: $properties;
54  }
```

Zaradi dodatne podpore pravilnega prikazovanja stilov za različne brskalnike je velikokrat potrebno vključiti ključne besede, katere pomenijo za vsak klic kar nekaj odvečnih vrstic kode. SASS[4] nam omogoča pripravo t.i. `mixinov`, s pomočjo katerih lahko pogosto uporabljene ključne besede pretvorimo v funkcijo, katero lahko nato v ostalih datotekah kličemo z ukazom `@include IMEFUNKCIJE`.

Primer uporabe takšnega klica:

```
1 .whiteBox {
2
3   ...
4
5   @include border-radius($border-radius);
6 }
```

5.2.2 Pogledi

Pogledi v ogrodju AngularJS[7] nam omogočajo preprosto delo s spremenljivkami in različnimi funkcijami. V podpoglavjih omenimo nekaj zanimivosti pri kreiranju različnih pogledov.

5.2.2.1 Priklapljanje nadzornika

Ko definiramo nov nadzornik, ga je potrebno priklopiti na posamezen pogled. to lahko v ogrodju AngularJS[7] storimo s pomočjo ključne besede `ng-controller`. Ko povežemo pogled z nadzornikom, lahko znotraj posameznega pogleda dostopamo do vseh spremenljivk znotraj obsega (`scope`) priključenega nadzornika.

Primer kode za priklapljanje nadzornika:

```
1 <div class="addList" ng-controller="AddlistCtrl">
2   ...
3 </div>
```

5.2.2.2 Delo z obrazci

Ogrodje AngularJS[7] nam omogoča poenostavljeno delo z obrazci. Posamezen obrazec lahko priklopimo na funkcijo v nadzorniku s pomočjo ključne besede `ng-submit`. Ključna beseda pomeni, da se bo ob sproženju gumba tipa `submit` - `(<button type='submit' ...>)` sprožila funkcija, ki je določena

znotraj obsega ključnega nadzornika.

Primer kode za ključno besedo `ng-submit`:

```
<form name="addListForm" ng-submit="addList()">

  ...

  <button type="submit">Create new list</button>
</form>
```

Ker moramo iz posameznega obrazca prebrati vnosna polja, ogrodje AngularJS poskrbi za vezavo s pomočjo ključne besede `ng-model`. Ko v posamezno vnosno polje vnesemo znake, nam AngularJS shrani vrednost vnosnega polja v definirano ime modela.

Na spodnjem primeru je razvidno, da s pomočjo vnosnega polja kreiramo objekt tipa:

```
list = {
  name: "",

  ...

}
```

Primer kode za delo s ključno besedo `ng-model`:

```
<form name="addListForm" ng-submit="addList()">
  <div class="form-group">
    <input type="text" class="form-control" id="name" name
       ="name" ng-model="list.name" placeholder="List
       title" autofocus required>
  </div>

  ...

```

```
8 </form>
```

V ogrodju AngularJS[7] je možna preprosta validacija obrazcov. S pomočjo ključne besede `ng-show` lahko skrijemo posamezen element, dokler ne zadošča določenim pogojem. Ti pogoji so shranjeni v spremenljivko, katero AngularJS za obrazce kreira avtomatsko. Tako lahko preprosto dodamo nov bločni element, kateremu dodamo ključno besedo `ng-show`, v njem pa definiramo pogoje.

Na spodnjem primeru je razvidno, da validacijo ustvarimo s tremi preprostimi koraki:

1. željenemu vnosnemu polju dodamo ključno besedo `required`
2. Dodamo nov bločni element, kateremu dodamo ključno besedo `ng-show`
3. Znotraj bločnega elementa ustvarimo pogoje, kdaj se naj opozorilno sporočilo prikaže.

Primer kode za validacijo forme s ključno besedo `ng-show` in ključno besedo `required`:

```
1  <form name="addListForm" ng-submit="addList()">
2
3  ...
4
5  <input type="text" class="form-control" id="name" name="
       name" ng-model="list.name" placeholder="List title"
       required>
6  <span ng-show="addListForm.$dirty &&
       addListForm.name.$error.required" class="help-block">We
        need a list name.</span>
7
8  ...
9
10 </form>
```

Pri zgornjem primeru opazimo, da pri ključni besedi `ng-show` konkateniramo dva pogoja v enega z logičnim operaterjem `IN (&&)`.

Prvi pogoj, `addListForm.$dirty` dobi vrednost resnično (`true`) takrat, ko smo aktivirali katerokoli vnosno polje znotraj posameznega obrazca. To storimo zato, da se izognemo prikazu opozorilnih sporočil pred tem, da uporabnik v obrazec začne vnašati podatke.

Drugi pogoj, `addListForm.name.$error.required`, pa dobi vrednost resnično (`true`) takrat, ko ob sproženju gumba tipa `submit` (`(<button type='submit' ...>)`) vnosno polje z definiranim imenom (v tem primeru`name`) ni izpolnjeno.

5.2.2.3 Pogled za avtomatsko predlaganje seznamov glede na vnos iskalnih parametrov

Ker smo želeli uporabniku olajšati iskanje posameznih seznamov, smo mu omogočili avtomatsko predlaganje seznamov glede na iskalne parametre znotraj vnosnega polja.

Tako smo za potrebe predlaganja seznamov morali kreirati pogled, ki nam prikazuje rezultate glede na iskane parametre.

Primer kode za pogled avtoamtskega predlaganja seznamov:

```
1 <div class="autocomplete col-xs-12 animation" ng-hide="
    autocompleteHidden">
2   <div class="item" ng-repeat="result in autocompleteQuery
      ">
3     <a class="row" ui-sref="list({ 'listHash':
        result.qr_hash })">
4       <div class="col-xs-1">
5         <img ng-src="{{ result.header_img_url }}" class="
            img img-responsive img-thumbnail"/>
6       </div>
7       <div class="col-xs-6">
```

```
8           <div class="title">{{ result.name }}<br></div>
9           <div class="description">{{ result.description |
                limitTo:70 }}...</div>
10        </div>
11        <div class="col-xs-5">
12          <span class="btn btn-light-pink pull-right" ui-
                sref="list({ 'listHash': result.qr_hash })">
13        View list
14        </span>
15      </div>
16      </a>
17    </div>
18
19    <div class="noresults" ng-show="autocompleteQuery.length
          == 0">
20    No results
21    </div>
22  </div>
```

Na tem mestu lahko omenimo še ključno besedo `ng-repeat`. Ključna beseda nam v pogledih omogoča preprosto iteracijo čez zbirke. Tako v zgorji kodi iteriramo čez zbirko `autocompleteQuery`, katere vrednost posamezne iteracije pripnemo v spremenljivko `result`.

Razvidno je tudi, da v primeru, ko nam iskanje vrne zbirko dolžine 0, prikažemo obvestilo 'No results'.

5.2.3 Storitve

5.2.3.1 Storitev REST[18]

Ker se nam klici na zaledni sistem ponavljajo, je bilo smiselno uporabiti storitev, ki nam olajša delo s klici Restangular[8]. Ker smo želeli klice in uporabo te storitve poenotiti, smo pripravili še eno storitev, ki upravlja z Restangular storitvijo.

Primer nekaj klicev na zaledni sistem s pomočjo pripravljene storitve:

```
1  angular.module('sharelyWebAppApp')
2  .factory('RestService', function ($http, Restangular,
       Session, HelperService) {
3      var service = {
4
5          searchQuery: function (search_param){
6              return Restangular.all('lists/').customGET('',
                   {search:search_param},
                   service._get_headers());
7          },
8          searchQueryPage: function (search_param, pageNum){
9              return Restangular.all('lists/').customGET('?
                   page='+pageNum, {search:search_param},
                   service._get_headers());
10         },
11         addNewList: function (listData){
12             return Restangular.all('lists/').post(listData
                   , {}, service._get_headers());
13         },
14         getList: function (listHash){
15             return Restangular.all('lists/').customGET(
                   listHash+'/', {}, service._get_headers());
16         },
17         updateList: function (listDATA){
18
19             var qrHash = listDATA.qr_hash + '/';
20             var endpoint = HelperService.getListTypeURL(
                   listDATA);
21
22             return Restangular.one(endpoint).customPUT(
                   listDATA, qrHash, '', service._get_headers
                   ());
23         },
24         deleteList: function (listDATA){
25
26             var qrHash = listDATA.qr_hash + '/';
```

```
27              var endpoint =
                    HelperService.getListTypeURLAlternative(
                    listDATA);
28
29              return Restangular.all(endpoint).customDELETE(
                    qrHash, {}, service._get_headers());
30          },
31
32          ...
33
34      };
35
36      return service;
37  });
```

5.2.3.2 Storitev za delo s pojavnimi okni

Za delo s pojavnimi okni uporabljamo storitev ngDialog[9]. Ker se več storitev ngDialog uporablja v različnih nadzornikih, smo pripravili dodatni storitev za delo z generičnimi pojavnimi okni.

Primer kode, ki nam prikaže potrditveno pojavno okno, s pomočjo katerega obvestimo uporabnika o možni nevarni akciji (npr. brisanje seznama):

```
1  angular.module('sharelyWebAppApp')
2  .service('DialogService', function (ngDialog) {
3
4      var dialogs = {};
5
6      dialogs.closeAll = function () {
7          return ngDialog.closeAll();
8      };
9
10     dialogs.destroyActionDialog = function () {
11         return ngDialog.openConfirm({
12             template: 'views/destroyDialog.html'
13         });
```

```
14      };
15
16      ...
17
18     return dialogs;
19 });
```

5.2.4 Nadzorniki

Za posamezne funkcionalnosti aplikacije nam v ogrodju AngularJS[7] skrbijo različni nadzorniki. V podpoglavjih omenimo nekaj zanimivih nadzornikov.

5.2.4.1 Dodajanje seznama

Pri dodajanju novega seznama je bilo potrebno pripraviti klic, ki nam doda nov seznam s parametri kot entiteto v zaledni sistem.

```
1  angular.module('sharelyWebAppApp')
2  .controller('AddlistCtrl', function ($scope, $state,
       ngDialog, RestService, toaster, HelperService) {
3
4      $scope.list ={
5          "name": "",
6          "is_private": false,
7          "is_invisible": false,
8          "description": "",
9          "header_img_url": null,
10         "list_categories": [],
11         "tags": []
12     };
13
14     $scope.addList = function () {
15
16         var addListFail = function (error) {
17             toaster.pop(TOAST_CONSTANTS.error,
                   RESPONSE_CONSTANTS.error,
                   RESPONSE_CONSTANTS.tryAgain);
```

```
18          };
19
20          var addListSuccess = function (success) {
21              ngDialog.closeAll();
22
23              var list = success.plain();
24
25              //On success redirect to new list
26              var qr_hash = list.qr_hash;
27              var listtype = HelperService.getListTypeState(
                    list);
28
29              $state.go(listtype, { 'listHash': qr_hash })
30          };
31
32          RestService.addNewList($scope.list).then(
                addListSuccess, addListFail);
33      }
34  });
```

Če si podrobneje pogledamo izvorno kodo, lahko opazimo na koncu klica `RestService.addNewList...` klic funkcije `.then()`. To pomeni, da s pomočjo t.i. obljube, katero vrne storitev RestService asinhrono obdelujemo rezultate klica. Če vrne klic uspeh, s pomočjo funkcije `.plain()` izločimo vse odvečne parametre in preusmerimo trenutno stran na novo dodan seznam.

5.2.4.2 Dodajanje novega predmeta posameznega seznama

Ko želi uporabnik dodati nov predmet v svoj večopravilen seznam, jo lahko doda s pritiskom na gumb "Dodaj nov predmet", ali pa s pomočjo posebnega ukaza znotraj vnosnega polja.

Če uporabnik prvi besedi v vnosnem polju (do prvega presledka) doda znak `#`, nam aplikacija doda novo kategorijo in karkoli za presledkom kot nov predmet.

Primer uporabe posebnega ukaza bi bil `#IMEKATEGORIJE IMEPREDMETA`.

To omogočimo s pomočjo sledeče kode:

```
1  var splitLine = itemTitle.split(" ");
2
3  if (splitLine.length > 0) {
4
5      if (splitLine[0].charAt(0) == "#") {
6
7          //shift returns and pops the first item
8          var categoryName = splitLine.shift().substr(1);
9
10         var categoryPostFail = function (error) {
11             toaster.pop(TOAST_CONSTANTS.error,
                   RESPONSE_CONSTANTS.error,
                   RESPONSE_CONSTANTS.tryAgain);
12         };
13
14         var categoryPostSuccess = function (catID) {
15
16             var itemTitle = splitLine.join(" ");
17             var category = returnCategoryForCatID(catID);
18
19             postNewItem(category, itemTitle);
20         };
21
22         $scope.addListCategory(categoryName).then(
               categoryPostSuccess, categoryPostFail);
23
24     }
25     else {
26
27         var category = returnCategoryForCatIndex(
               $scope.currentlySelectedItemCategory);
28
29         if (category != null) {
30             postNewItem(category, itemTitle);
```

```
31          }
32      }
33  }
```

```
1  var postNewItem = function (category, itemTitle) {
2
3    var itemForCategory = {
4        "id": category.list_items.length + 1,
5        "name": itemTitle,
6        "order": 0,
7        "color_type": "color-one",
8        "description": "",
9        "category": category.id
10   };
11
12
13   if (itemTitle != "" && itemTitle != null) {
14
15       var itemPostFail = function (error) {
16
17       };
18
19       var itemPostSuccess = function (success) {
20
21           //Change to the correct response ID (for
                itemForCategory)
22           itemForCategory.id = success.id;
23
24           category.list_items.splice(0, 0, itemForCategory
                );
25
26           category.list_items = reindexItems(
                category.list_items);
27
28           $scope.newItemInput = "";
29
30       };
31
```

```
32        RestService.setItemForCategoryForList(
              $scope.list.qr_hash, category.id, itemForCategory
              ).then(itemPostSuccess, itemPostFail);
33
34    }
35 };
```

```
1  var reindexItems = function (list_items) {
2
3      //update all item indexes
4      for (var i = 0; i < list_items.length; i++) {
5          var item = list_items[i];
6
7          item.order = i;
8      }
9
10     return list_items;
11 }
```

5.2.5 Nadzornik za avtomatsko predlaganje seznamov glede na vnos iskalnih parametrov

Ker smo želeli uporabniku olajšati iskanje posameznih seznamov, smo mu omogočili avtomatsko predlaganje seznamov glede na iskalne parametre znotraj vnosnega polja.

To smo storili s pomočjo sledeče kode:

```
1  $scope.autocompleteHidden = true;
2
3  $scope.typingStarted = function () {
4
5    $scope.autocompleteHidden = false;
6
7    var searchQuery = $scope.searchInput;
8
```

```
9     if (searchQuery == "") {
10      $scope.autocompleteHidden = true;
11    }
12
13    /*
14    Search submit success error func
15    */
16
17    var searchQuerySubmitSuccess = function (success) {
18      $scope.autocompleteQuery = success.results
19    };
20
21    var searchQuerySubmitFail = function (error) {
22      $scope.showErrorMessage = true;
23    };
24
25    RestService.searchQuery(searchQuery).then(
          searchQuerySubmitSuccess, searchQuerySubmitFail);
26  };
```

Iz kode je razvidno, da polje za predlaganje privzeto skrijemo s pomočjo spremenljivke `autocompleteHidden`. Celotno vnosno polje vežemo na ključno besedo (znotraj elementa v pogledu) `ng-change='typingStarted()'`, ki se sproži ob vsaki spremembi (npr. vnosu znakov) v vnosnem polju. Naknadno je bilo potrebno preverjati, če je uporabnik nekaj vnesel in nato vnešeno izbrisal. Če se to zgodi, je potrebno avtomatsko predlaganje skriti. To lahko v zgornji kodi razberemo v kodi, kjer se začne pogojni stavek `if (searchQuery == '')`.

V primeru, da uporabnik začne karkoli tipkati, se torej sproži klic na zaledni sistem, ki nam vrne rezultate glede na iskane parametre, bodisi po posameznih značkah, ali pa po imenu seznama.

5.2.6 Nastavitve strežnika uporabniškega vmesnika

Kot smo že omenili, za serviranje datotek uporabljamo nginx[20]. Ker imamo ločene aplikacije uporabniškega vmesnika in zalednega sistema, moramo posledično spisati dve nastavitveni datoteki za strežniški sistem nginx.

Z novo dobo interneta so se pojavili tudi novi standardi. Včasih so podjetja dajala veliko težo na poddomensko ime `www`. Sedaj, ko je že večini ljudi jasno, da v brskalniku naslavljajo spletno stran, postaja končnica `www` redundantna. Zato v naši aplikaciji zahtevke `www` še vseeno sprejemamo, ampak jih s pomočjo posebnega zahtevka preusmerjamo na naslov brez poddomene.

Če bi uporabnik torej vtipkal `http://www.spletnastran.com`, bi ga naša konfiguracijska datoteka avtomatsko preusmerila na `http://spletnastran.com`.

```
1 server {
2         listen              80;
3         client_max_body_size 4G;
4         server_name XYZ.org;
5         keepalive_timeout 5;
6
7         root /home/sharely/sharelyWebApp/releases/current/
              sharelyWebApp/;
8
9         gzip              on;
10        gzip_min_length   1000;
11        gzip_proxied      expired no-cache no-store private
              auth;
12        gzip_types text/plain text/css application/json
              application/x-javascript text/xml application/
              xml text/javascript;
13        gzip_disable      "MSIE [1-6]\.";
14
15        location / {
16            try_files $uri /index.html;
17        }
18
19 }
```

```
20
21 server {
22          listen                80;
23          server_name "~^www\.(.*)$" ;
24          return 301 $scheme://$1$request_uri ;
25 }
```

Poglavje 6

Sklepne ugotovitve

V sklopu diplomske naloge smo razvili aplikacijo za deljenje večopravilnostnih seznamov, katera je primerna tako za končnega uporabnika, kot tudi za manjša podjetja. Cilj aplikacije je bil uporabniku olajšati pisanje večopravilnostnih seznamov. Aplikacija omogoča dodajanje, deljenje, kopiranje, brisanje, všečkanje in pregled večopravilnostnih seznamov. Vse naštete akcije so možne preko intuitivnega grafičnega vmesnika, ki je bil izdelan posebej za namene uporabe aplikacije. Za razvoj zalednega sistema smo uporabili tehnologije, kot so Python[13], Django[17], Django Rest Framework[19], REST[**?**] in nginx[20], za razvoj uporabniškega vmesnika pa tehnologije, kot so AngularJS[7], SASS[4], HTML5[2] in Javascript[5].

Pri razvoju aplikacije na večje težave nismo naleteli, bilo pa je potrebnega nekaj dodatnega prilagajanja za različne širine zaslonov, kot tudi za pravilen prikaz na različnih brskalnikih.

Aplikacija je še v zgodnji fazi razvoja in bo v prihodnosti doživela še nekaj tako lepotnih, kot tudi funkcionalnih popravkov in dodatkov. Dolgoročni načrt za razvoj aplikacije je tako izdelava mobilne aplikacije za sistem iOS in Android, kot tudi večjezična podpora.

Literatura

[1] HTML. [Online]. Dosegljivo: http://en.wikipedia.org/wiki/HTML. [Dostopano 25. 6. 2015].

[2] HTML5. [Online]. Dosegljivo: http://en.wikipedia.org/wiki/HTML5. [Dostopano 25. 6. 2015].

[3] CSS. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Cascading_Style_Sheets. [Dostopano 25. 6. 2015].

[4] SASS. [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Sass_(stylesheet_language). [Dostopano 25. 6. 2015].

[5] Javascript. [Online]. Dosegljivo: https://en.wikipedia.org/?title=JavaScript. [Dostopano 25. 6. 2015].

[6] Bootstrap. [Online]. Dosegljivo: http://getbootstrap.com. [Dostopano 25. 6. 2015].

[7] AngularJS. [Online]. Dosegljivo: https://angularjs.org. [Dostopano 25. 6. 2015].

[8] Restangular. [Online]. Dosegljivo: https://github.com/mgonto/restangular. [Dostopano 29. 6. 2015].

[9] ngDialog. [Online]. Dosegljivo: https://github.com/likeastore/ngDialog. [Dostopano 29. 6. 2015].

[10] Yeoman. [Online]. Dosegljivo:
http://yeoman.io. [Dostopano 25. 6. 2015].

[11] Grunt. [Online]. Dosegljivo:
http://gruntjs.com. [Dostopano 25. 6. 2015].

[12] Bower. [Online]. Dosegljivo:
http://bower.io. [Dostopano 25. 6. 2015].

[13] Python. [Online]. Dosegljivo:
https://www.python.org. [Dostopano 25. 6. 2015].

[14] PIL. [Online]. Dosegljivo:
http://www.pythonware.com/products/pil/. [Dostopano 25. 6. 2015].

[15] The NIST Definition of Cloud Computing. [Online]. Dosegljivo:
http://faculty.winthrop.edu/domanm/csci411/Handouts/NIST.pdf.
[Dostopano 29. 6. 2015].

[16] A view of Cloud Computing. [Online]. Dosegljivo:
http://delivery.acm.org/10.1145/1730000/1721672/p50-
armbrust.pdf?ip=94.140.95.216&id=1721672&acc=OPEN&key=
4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35
%2E6D218144511F3437&CFID=690681284&CFTOKEN=31975543
&__acm__=1436189525_65c98b888c1378d87b363feb0ebd6659. [Dostopano 6. 7. 2015].

[17] Django. [Online]. Dosegljivo:
https://www.djangoproject.com. [Dostopano 25. 6. 2015].

[18] REST. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Representational_state_transfer. [Dostopano 25. 6. 2015].

[19] Django Rest Framework. [Online]. Dosegljivo:
http://www.django-rest-framework.org. [Dostopano 25. 6. 2015].

[20] nginx. [Online]. Dosegljivo:
https://en.wikipedia.org/wiki/Nginx. [Dostopano 25. 6. 2015].

[21] nginx. [Online]. Dosegljivo:
http://httpd.apache.org. [Dostopano 25. 6. 2015].

[22] Apache vs nginx. [Online]. Dosegljivo:
https://www.digitalocean.com/community/tutorials/apache-vs-nginx-practical-considerations. [Dostopano 25. 6. 2015].

[23] Virtualenv. [Online]. Dosegljivo:
https://virtualenv.pypa.io/en/latest/. [Dostopano 25. 6. 2015].

[24] GIT. [Online]. Dosegljivo:
https://en.wikipedia.org/?title=Git_(software). [Dostopano 25. 6. 2015].

[25] Fabric. [Online]. Dosegljivo:
http://www.fabfile.org. [Dostopano 25. 6. 2015].

[26] PyCharm. [Online]. Dosegljivo:
https://www.jetbrains.com/pycharm/. [Dostopano 25. 6. 2015].

[27] Kombinatorika, variacije s ponavljanjem. [Online]. Dosegljivo:
https://sl.wikipedia.org/wiki/Kombinatorika#Variacije. [Dostopano 29. 6. 2015].

[28] L. Fortnow, "Viewpoint: Time for computer science to grow up", *Communications of the ACM*, št. 52, zv. 8, str. 33–35, 2009.

[29] D. E. Knuth, P. Bendix. "Simple word problems in universal algebras", v zborniku: Computational Problems in Abstract Algebra (ur. J. Leech), 1970, str. 263–297.

[30] L. Lamport. *LaTEX: A Document Preparation System*. Addison-Wesley, 1986.

[31] O. Patashnik (1998) BibTeXing. [Online]. Dosegljivo: http://ftp.univie.ac.at/packages/tex/biblio/bibtex/contrib/doc/btxdoc.pdf. [Dostopano 18. 9. 2014].

[32] Wunderlist, aplikacija. [Online]. Dosegljivo: https://www.wunderlist.com. [Dostopano 7. 7. 2015].

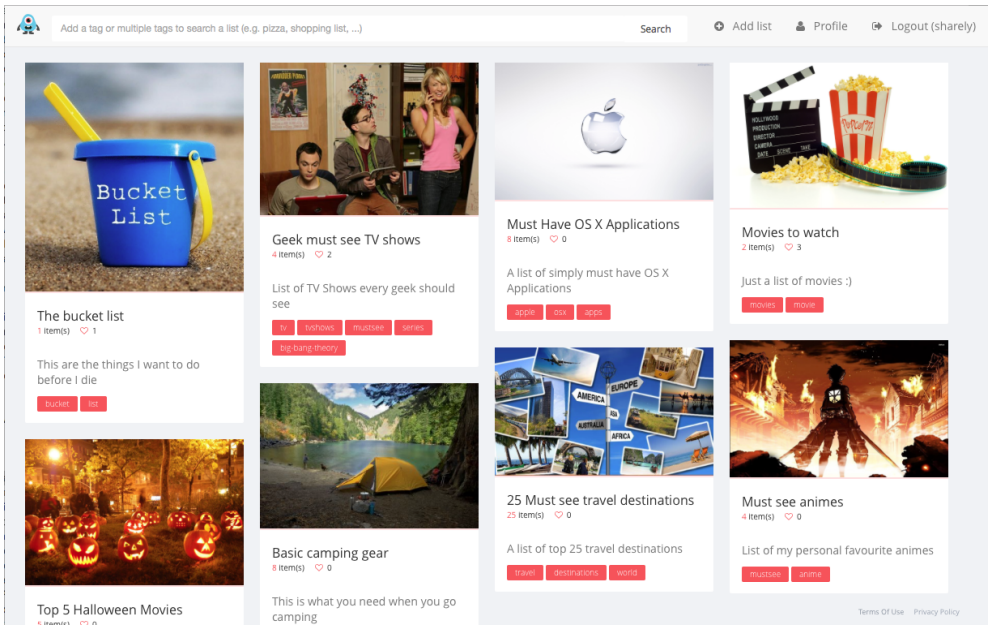
[33] Todoist, aplikacija. [Online]. Dosegljivo: https://todoist.com. [Dostopano 7. 7. 2015].

[34] ANY.DO, aplikacija. [Online]. Dosegljivo: http://www.any.do/anydo/. [Dostopano 7. 7. 2015].

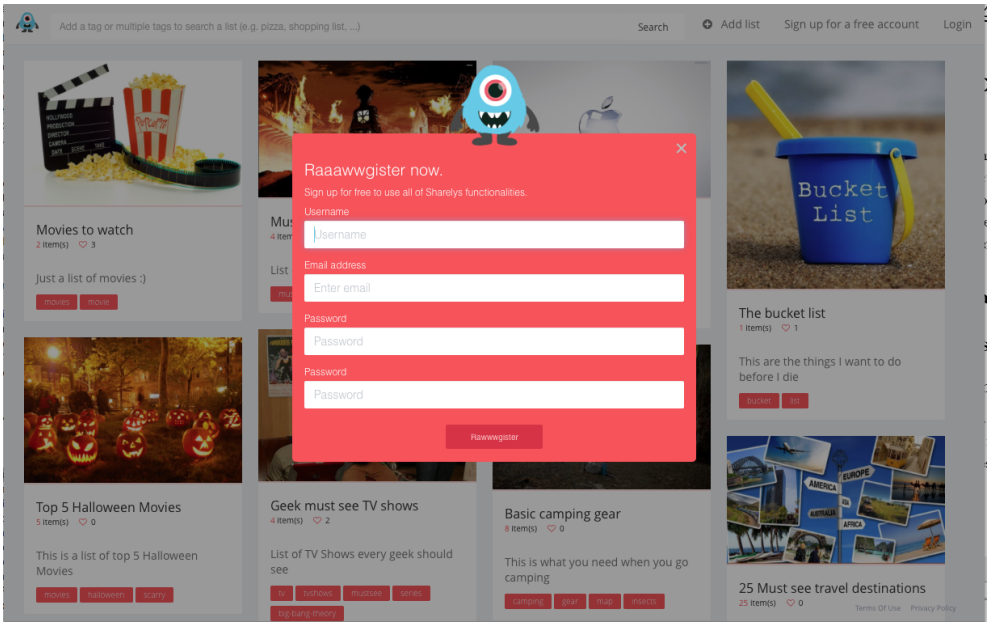
Dodatek A

Izgled uporabniškega vmesnika

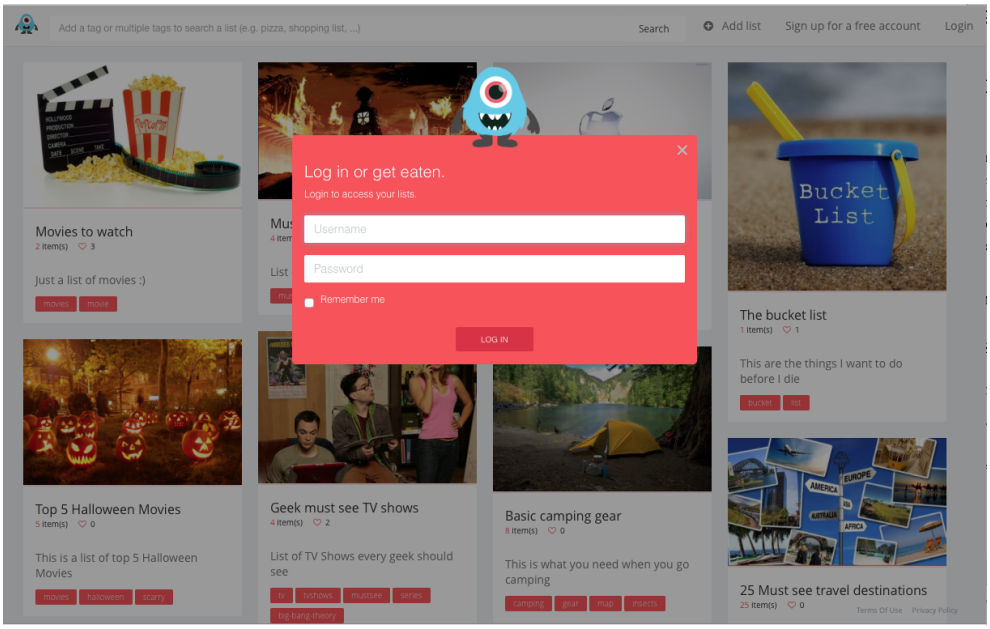
Priložene so predloge izgleda uporabniškega vmesnika.



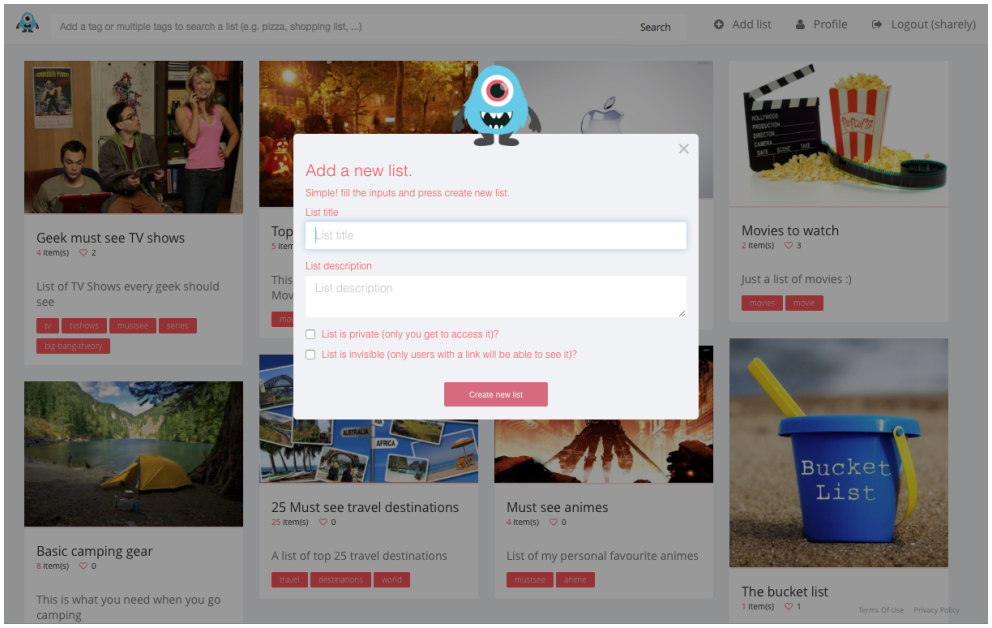
Slika A.1: Izgled pogleda - pristajalna stran



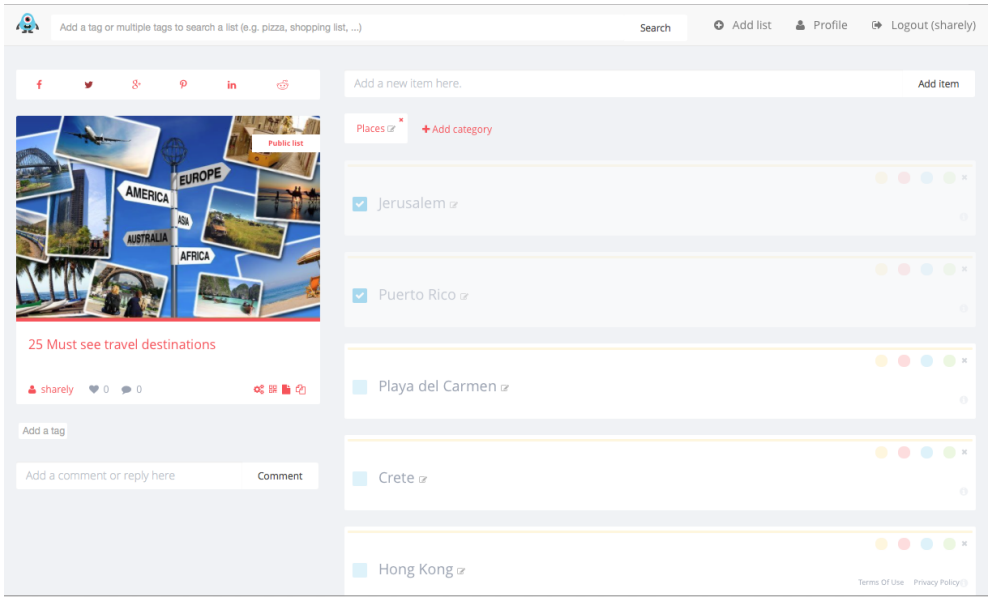
Slika A.2: Izgled pogleda - registracija uporabnika



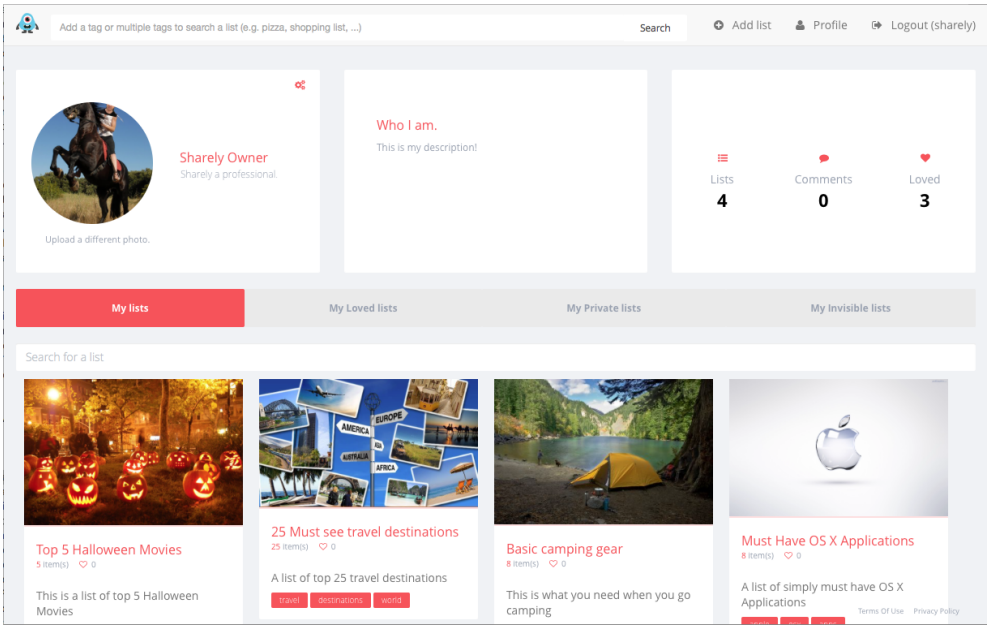
Slika A.3: Izgled pogleda - prijava uporabnika



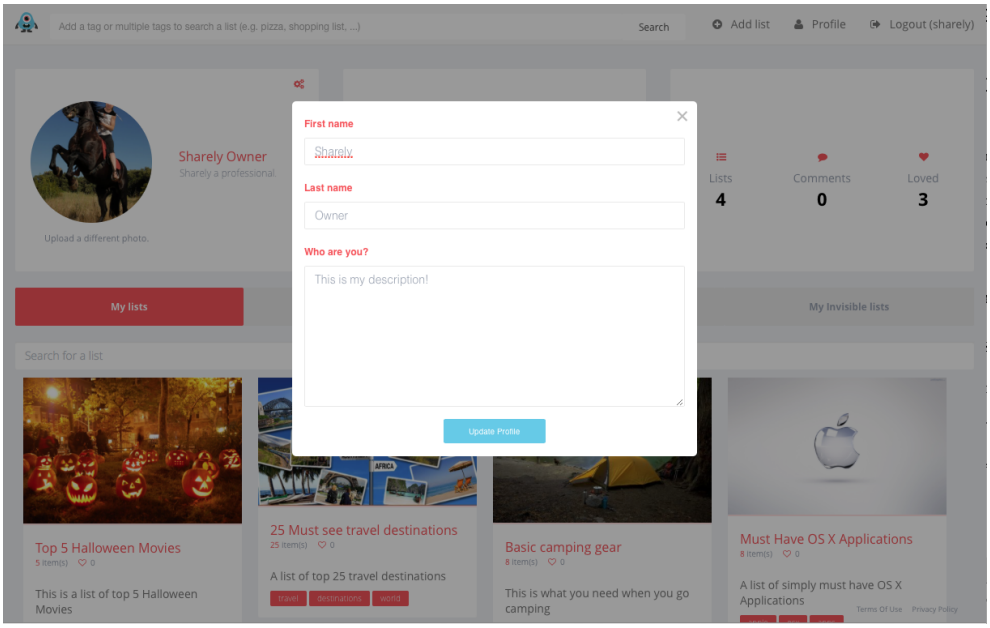
Slika A.4: Izgled pogleda - dodajanje novega seznama



Slika A.5: Izgled pogleda - podroben pregled seznama



Slika A.6: Izgled pogleda - profila uporabnika



Slika A.7: Izgled pogleda - urejanja profila uporabnika